به نام خدا

تمرین 4 طراحی زبان‌های برنامه‌سازی

جناب آقای دکتر ایزدی

سارا آذرنوش

98170668

(1

(value-of <<x>>

y = (2) , x = (7))
= (7)
(value-of 1
y = (2) , x = (7))
= (1)

(value-of <<-(x,1)>>

Y = (2) , x = (7))

= (6)

(value-of << x = -(x,1) in –(x,y)>>

Y = (2) , x = (7)) =
(4)
(value-of <<–(-(x,8) , y)>>
Y = (4) , y = (2), x = (7))

= (-5)

(value-of<< let y = let x = -(x,1) in -(x,y) in -(-(x,8),y)>>

Y = (2) , x = (7))
= (-5)

(value-of <<let y = 2 in let y = let x = -(x,1) in -(x,y) in -(-(x,8),y)>>

X = (7))
= (-5)

(value-of <<let x = 7 in let y = 2 in let y = let x = -(x,1) in -(x,y) in –(-(x,8),y)>>
)(-5)

(2

Environment و سایر مانند بقیه است قوانین استنتاج، گرامر و مقادیر آن را اضافه میکنیم.

(Value-of pow-exp (exp1 exp2)) = (num-val (exp num1 num2))

(value-of exp1 env) = Val1 , (value-of exp2 env) = Val2

(expval->num val1) = num1 , (expval->num val2) = num2

```
(define grammar
  '((program (expression) a-program)
    (expression
(number) const-exp)
    (expression
("^" "(" expression "," expression ")") pow-exp)
;other grammers
     ))

(define value-of
  (lambda (exp env)
    (cases expression exp
      (const-exp (num) (num-val num))
      (pow-exp (exp1 exp2)
        (let ((val1 (value-of exp1 env))
              (val2 (value-of exp2 env)))
          (let ((num1 (expval->num val1))
                (num2 (expval->num val2)))
            (num-val (exp num1 num2)))))
;other expressions
))
```

3)

محیط همان است مقادیر جدید در گرامر و value اضافه میکنیم.

(value-of emptylist-exp exp1) = (emptylist-val)

(value-of car-exp exp1) = (car val1)

(Value-of exp1) = val1

(value-of cdr-exp exp1) = (cdr val1)

(Value-of exp1) = val1

(value-of null?-exp exp1) = emptylist? : #t : #f

(Value-of exp1) = val1

(value-of concatenate -exp exp1 exp2) = (append val1 val2)

(Value-of exp1) = val1 , (Value-of exp2) = va2

(value-of sort-exp exp1) = (sort val1)

(Value-of exp1) = val1

```
(define grammar
  '((program (expression) a-program)
    (expression
     ("emptylist") emptylist-exp)
    (expression
     ("car" "(" expression ")") car-exp)
    (expression
      ("cdr" "(" expression ")") cdr-exp)
```

```
    (expression
     ("null?" "(" expression ")") null?-exp)
    (expression
     ("concatenate" "(" expression "," expression ")") concatenate-exp)
    (expression
     ("sort" "(" expression ")") sort-exp)
    ))
(define value-of
  (lambda (exp env)
    (cases expression exp
      (const-exp (num) (num-val num))
      (emptylist-exp ()
        (emptylist-val))
      (car-exp (exp1)
        (let ((val1 (value-of exp1 env)))
          (expval->car val1)))
      (cdr-exp (exp1)
        (let ((val1 (value-of exp1 env)))
          (expval->cdr val1)))
      (null?-exp (exp1)
        (let ((val1 (value-of exp1 env)))
          (let ((bool1 (expval->emptylist? val1)))
            (bool-val bool1))))
      (concatenate-exp (exp1 exp2)
        (let ((val1 (value-of exp1 env))
              (val2 (value-of exp2 env)))
            (list-val (append val1 val2))))
       (sort-exp (exp1)
```

```
        (let ((val1 (value-of exp1 env)))
          (list-val (sort val1))))
      )))

(define expval->car
  (lambda (val)
    (cases expval val
      (cons-val (first rest) first)
      (else error ))))

(define expval->cdr
  (lambda (val)
    (cases expval val
      (cons-val (first rest) rest)
      (else error))))

(define expval->emptylist?
  (lambda (val)
    (cases expval val
      (emptylist-val () #t)
      (cons-val (first rest) #f)
      (else error))))
```

4)

یک ورودی را چند ورودی میکنیم و مقادیر جدید را در گرامر و ولیو اضافه میکنیم.

```
(value-of (letc-exp vars exps body) env) =
(value-of body env) if null? var1 :
        (value-of (letc-exp (cdr vars) (cdr exps) body)
                (extend-env (car vars) (value-of (car exps) env))
```

```
(define grammar
  '((program (expression) a-program)
    (expression ("letc"(identifier "=" expression) "in" expression) letc-exp)))

(define value-of
  (lambda (exp env)
    (cases expression exp
      (const-exp (num) (num-val num))
        (letc-exp (varl expl body)
                (if (null? varl)
                    (value-of body env)
                    (let ((var1 (car var-list))
                        (val1 (value-of (car expl) env)))
                      (value-of (letc-exp (cdr varl)(cdr expl)body)
                            (extend-env var1 val1 env)))))
      ; other expressions
      )))
```

5)

مانند سوال امتحان میشود.

در اینجا n =5 ورودی دادیم و فیبوناتچی 5 را محاسبه میکند.

```
let f = proc (maker)
        proc (x)
                if zero? (x)
                then 0
                else if zero? (-(x, 1))
                then 1
        else -(((maker maker) -(x, 1)), -(0, ((maker maker) -(x, 2))))
in let main = proc (x) ((f f) x)
        in (main 5)
```

6)

مانند سوال 4 است با یک ورودی بیشتر و حالت letrec

(value-of (letrecc-exp p − names p − vars p − exps p − bodies body) env)=

(value-of body ρ) if null?p – names :

(value-of (letrecc-exp (cdr p − names)(cdr p − vars)(cdr p − bodies) body)

(extend-env-rec (car p − names) car p − vars) car p − bodies) env)))

```
(define grammar
  '((program (expression) a-program)
    (expression ("letrecc"(identifier "(" (identifier ",") ")" "=" expression) "in" expression) letrecc-exp)))

(define value-of
  (lambda (exp env)
    (cases expression exp
      (letrecc-exp (p-names b-vars p-bodies body)
                   (if (null? p-name)
                     (value-of body env)
                     (let ((var1 (car p-name))
                           (val1 (car b-vars))
                           (vab (car p-bodies)))
                       (value-of (letrecc-exp (cdr p-names) (cdr b-vars) (cdr p-bodies) body)
                                 (extend-env var1 val1 vab env)))))
      ; other expressions
      )))
```